**PCT** WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04L 12/56 — A2

(11) International Publication Number: **WO 99/59303**

(43) International Publication Date: 18 November 1999 (18.11.99)

(21) International Application Number: PCT/SE99/00788

(22) International Filing Date: 11 May 1999 (11.05.99)

(30) Priority Data:
9801744-5 14 May 1998 (14.05.98) SE

(71) Applicant *(for all designated States except US)*: TELIA AB (publ) [SE/SE]; Mårbackagatan 11, S-123 86 Farsta (SE).

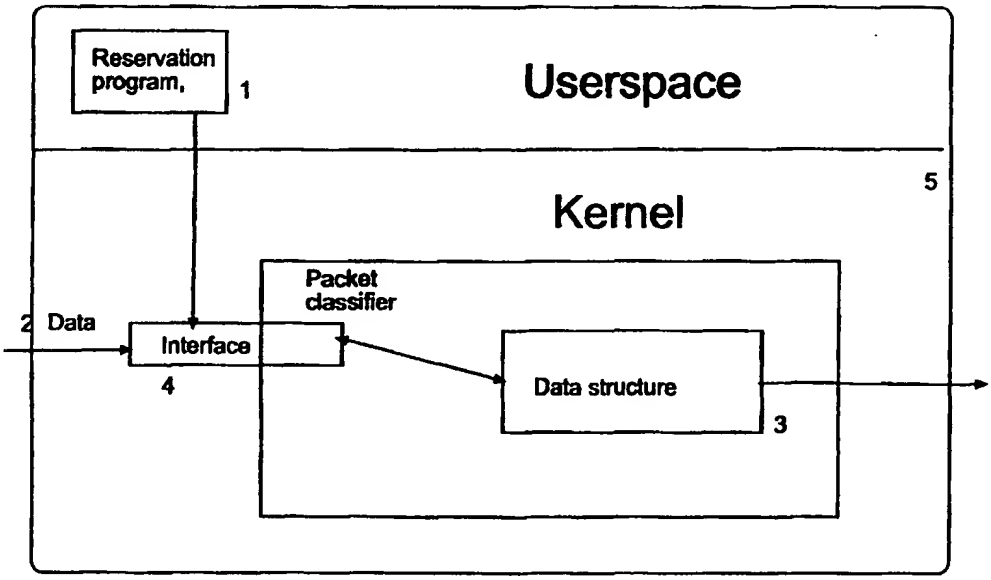
(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: BORG, Niklas [SE/SE]; Kårhusvägen 4, S-977 54 Luleå (SE). FLODIN, Malin [SE/SE]; Docentvägen 273, S-977 52 Luleå (SE).

(74) Agent: PRAGSTEN, Rolf; Telia Research AB, Vitsandsgatan 9, S-123 86 Farsta (SE).

(81) Designated States: EE, LT, LV, NO, US, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

**Published**
*Without international search report and to be republished upon receipt of that report.*

(54) Title: A COMMUNICATIONS NETWORK OR AN IP-NETWORK WHICH INCORPORATES A PACKET CLASSIFIER




(57) Abstract

A communications network or an IP-network which incorporates a packet classifier in an end system or a router which inspects each incomming packet based on information in the packet headers and determines how each packet should be treated. Said packet classifier divides packet into classes when a large number of filters are stored.

*FOR THE PURPOSES OF INFORMATION ONLY*

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| AL | Albania | ES | Spain | LS | Lesotho | SI | Slovenia |
| AM | Armenia | FI | Finland | LT | Lithuania | SK | Slovakia |
| AT | Austria | FR | France | LU | Luxembourg | SN | Senegal |
| AU | Australia | GA | Gabon | LV | Latvia | SZ | Swaziland |
| AZ | Azerbaijan | GB | United Kingdom | MC | Monaco | TD | Chad |
| BA | Bosnia and Herzegovina | GE | Georgia | MD | Republic of Moldova | TG | Togo |
| BB | Barbados | GH | Ghana | MG | Madagascar | TJ | Tajikistan |
| BE | Belgium | GN | Guinea | MK | The former Yugoslav Republic of Macedonia | TM | Turkmenistan |
| BF | Burkina Faso | GR | Greece | | | TR | Turkey |
| BG | Bulgaria | HU | Hungary | ML | Mali | TT | Trinidad and Tobago |
| BJ | Benin | IE | Ireland | MN | Mongolia | UA | Ukraine |
| BR | Brazil | IL | Israel | MR | Mauritania | UG | Uganda |
| BY | Belarus | IS | Iceland | MW | Malawi | US | United States of America |
| CA | Canada | IT | Italy | MX | Mexico | UZ | Uzbekistan |
| CF | Central African Republic | JP | Japan | NE | Niger | VN | Viet Nam |
| CG | Congo | KE | Kenya | NL | Netherlands | YU | Yugoslavia |
| CH | Switzerland | KG | Kyrgyzstan | NO | Norway | ZW | Zimbabwe |
| CI | Côte d'Ivoire | KP | Democratic People's Republic of Korea | NZ | New Zealand | | |
| CM | Cameroon | | | PL | Poland | | |
| CN | China | KR | Republic of Korea | PT | Portugal | | |
| CU | Cuba | KZ | Kazakstan | RO | Romania | | |
| CZ | Czech Republic | LC | Saint Lucia | RU | Russian Federation | | |
| DE | Germany | LI | Liechtenstein | SD | Sudan | | |
| DK | Denmark | LK | Sri Lanka | SE | Sweden | | |
| EE | Estonia | LR | Liberia | SG | Singapore | | |

Title.

A communicationsnetwork or a IP-network which incorporates a packet classifier.

Technical field.

A communications network or a IP-network which incorporates a packet classifier in a end system or a router which inspects each incoming packet based on information in the packet headers and determines how each packet should be treated.

State of the art.

A packet classifier is a program in an end system or a router that inspects each incoming packet and based on the information in the packet headers determines how each packet should be treated. Thus, a packet classifier divides packets into different classes.

An operating system can take advantage of a packet classifier in many different ways. First, a packet classifier in an end system can decide which process to wake up to take care of the packets (e.g., start a user level protocol stack: D. R. Engler and M. F. Kaashoek: *DPF: Fast, Flexible Message Demultiplexing using Dynamic Code Generation*. Proc. of ACM SIGCOMM'96, August 1996.).

Second, a router can use a packet classifier to make the management of packet flows more efficient by marking packets. This makes them easier to process when sent to the next hop router, so-called tag or IP switching (Y. Rekhter, B. Davie, D. Katz, E. Rosen, G. Swallow and D. Farinacci: *Tag Switching Architecture - Overview*. Internet Draft, July 1997. Work in progress.and (*IP Switching: The Intelligence of Routing, the Performance of Switching*. Ipsilon Networks white paper,

February 1996. http://www.ipsilon.com/productinfo/wp-ipswitch.html.).

Third, a packet classifier can be used in firewalls. For example, a company is not interested in any traffic to its local network with the exception of www-traffic to a specific computer, ftp traffic to another and PPP-connections to a third computer. These rules are inserted as filters in a packet classifier and the packets that match one of those criteria are the only ones that are allowed to pass the firewall.

Fourth, a packet classifier can be used in Quality of Service, QoS, solutions. Traditionally the Internet has only supported best effort service, no guarantees on delay or delay variation (jitter) has been given by the network. To offer different levels of service, packets must be classified so the scheduler can give each packet the requested service. Packet classification is done by examining a number of fields in the packet headers. Fields of interest are for example: source address, destination address, source port and destination port. These fields are compared with the filters stored in the packet classifier. With each filter is also an action stored. The action describes what to do if the incoming packet matches the filter. The appropriate action depends on the context in which the packet classifier is used.

Technical problem.

The invention seeks to provide a efficient packet classifier which can be implemented in applications of the above-mentioned kind.

The classifier is not intended to solve one single problem but is instead designed to solve a group of problems and

specially packet classification when a large amount of filters are stored.

Disclosure of the invention.

According to an aspect of the present invention, there is provided a Packet Classifier as specified in claim 1.

Advantageous effects of the invention.

A Packet Classifier formed in accordance with the invention has a number of advantages.

The packet classifiers data structure can scale well when additional filters are inserted, the filter specification is general enough to support classification on subnets etc. and the data structure is memory efficient. The advantages of the implemented packet classifier can be summarised as the following:

1.Scaleable data structure
2.General filter specifications
3.Low memory consumption

Embodiments of the invention will now be described, by way of example, with reference to the accompanying drawings, in which:

Fig. 1. illustrates, in schematic form, an packet classifier according to our invention and different parts of the classifier.

Fig. 2. illustrates, an IP and UDP header.

Fig. 3. illustrates, the binomial trees of the data structure.

Fig. 4. illustrates, the filter (SP=16, TOS=0) is the first filter inserted.

Fig. 5. illustrates, filters 0100 and 010x have been inserted.

Fig. 6. illustrates, the trie tree is extended with filters 01xx, 1000 and 1001.

The implemented packet classifier is shown in figure 1 and consists of three more or less detached parts. These are: the reservation program 1 which is a standalone program, the interface between the packet flows 2 and the data structure 3 and the data structure itself. The main part, the data structure 3 and the interface 4 between it and the packets, is implemented as a part of the kernel 5 in the operating system NetBSD. This is done to speed up the packet classification and to achieve access to the incoming packets in a natural way. The reservation program does not have the same demands for swiftness, and interaction with the incoming packets is not needed either. Therefore it is placed in user space.

The implementation of the packet classifier is relatively general. The classifier is not optimized to solve one single problem but is instead designed to solve a group of problems, namely packet classification when a large amount of filters are stored.

Filters

A filter consists of a reservation and a number of fields in the packet headers. A filter can consist of any combination of the six fields shown in figure 2.

The following abbreviations are introduced for these fields:

| | | | |
|---|---|---|---|
| • SA | - | Source Address | 32 bits |
| • DA | - | Destination Address | 32 bits |
| • SP | - | Source Port | 16 bits |
| • DP | - | Destination Port | 16 bits |
| • Prot | - | Protocol | 8 bits |
| • TOS | - | Type of Service | 8 bits |

The reason for choosing these field are that they can specify a flow. Some prior art uses the SA,DA,DP and Prot fields to specify flows and filters. Differential Services uses the TOS field in some proposals in the prior art.
At start up time it is possible to give different priority orders to the fields that shall be classified on (e.g., the packet classifier can be configured to give the source address field higher priority than the protocol field). A priority order between the different fields is necessary since an incoming packet can match more then one filter. For example, a packet can match one filter which is based on the source and destination ports and at the same time match another filter which is based on the source and destination address. The priority order then decides which filter that will be chosen.

Granularity of flows

An important quality for a fast packet classifier is that filters can match not only individual flows but also groups of flows. The property that indicates on which detail level a flow is specified is called granularity. With a coarser

granularity each single flow does not have to be specified when making the same filter for many flows, thus a complete subnet can be specified with just one filter. The advantage of using a coarser granularity is that many flows can share a single filter, meaning that the filters are fewer. This results in faster packet classification. The disadvantage is a lower level of detail. One flow cannot be separated from the other. Our packet classifier supports filter specifications on subnets. Source and destination addresses can have subnet filters made on bit level. The notation 131.115.22.11/29 means that a filter is made on the high order 29 bits in the address.

The reservation program

The reservation program is the part of the packet classifier where new filters are made and old ones are deleted. Filters consist of header fields and a corresponding reservation. Filters are inserted as the user enters the filters in a configuration file. After that the user compiles the configuration file with the reservation program. The reservation program gives, if the configuration file is syntactically correct, a compiled file back. This file is then copied by the user, via a device, down to kernel. Addition and removal of filters can be done at any time. No rebooting is necessary and the new filters are accessible for the classifier as soon as they are copied down to the kernel.

The interface between the incoming packets and the data structure

The interface forwards incoming packets to the data structure. It also takes care of the interrupts that are generated when new filters are copied down to the kernel. When these filters arrive it is the interface that takes care of them and inserts them into, or removes them from, the data structure.

The data structure

The filters and their reservations are stored in the data structure. This is the most important part of a fast packet classifier, at least when many filters are stored. The reason for this is that the cost for filter lookups in a linear structure (an array or a linked list) is proportional to the numbers of filters. This is not a problem if the packet classifier only needs to store a small number of filters at the same time. Since the number of filters is small, the lookup time will also be small. For a packet classifier intended to be used in a router handling QoS matters, the working conditions are totally different. The number of simultaneously stored filters can often be counted in 1000:s and then it is not appropriate to use a linear data structure. In this case it is important to find matching filters without having to traverse all of them. The faster we can find a match between the incoming packet and a filter the better. We have chosen to build our structure as a combination between binomial trees and tries. We have created a data structure with six binomial trees where each node in the binomial tree is a trie.

**Binomial structure**

How a filter is inserted in the data structure depends on the priority order between the different fields. If we assume that SA has the highest priority followed by DA, SP, DP, TOS and PROT, our data structure consists of six binomial trees structured as in figure 3. There are six trees where the first and the second tree follow the same binomial structure as the ones below. Please observe that only the four smallest trees are pictured in figure 3.

The first tree contains all filters associated with the highest priority field (i.e., if a filter includes the highest priority field, all fields of the filter will be inserted in the first tree). If the filter does not include this field but the second highest, all fields of the filter are inserted in the second tree etc. If DP had been given a higher priority than SP at the configuration, DP would be found in SP's position in figure 3 and vice versa.
These binomial trees are a bit more complex than ordinary binomial trees. In figure 3 the nodes represent smaller trees. These smaller trees are trie trees and will be described further down. When inserting a filter in this structure, we first insert the highest priority field in the trie tree at the top of the corresponding binomial tree. All nodes in a trie tree has pointers to its following binomial trees. When the high priority field has been inserted, the rest of the fields are inserted recursively in the following binomial trees pointed at by this newly inserted node. Accordingly, each node in a binomial tree points to multiple, smaller binomial trees. The resource reservation is stored in the node containing the last inserted field.

Figure 4 illustrates the third tree after inserting the filter (SP=16, TOS=0). The node in the trie tree contains pointers to the following binomial trees. In this example the same priority order as mentioned above (SA, DA, SP, DP, TOS, PROT) is assumed.

If the packet classifier is used in a router, the router must classify every incoming packet to find out if there are any specific resources allocated for it. This is done by searching the filters to find one that matches the incoming packet. The search starts in the first binomial tree. If a

matching filter is found in this tree, the associated action is performed, otherwise the search continues in the second tree etc. If no filter is found in any of the six trees, then a default action is performed.

**Trie tree structure**

The small trees appearing as nodes in figure 3 are trie trees. The nodes in these small trees are grouped in such a way that when a lookup is performed and a node that does not match the incoming packet is found, the lookup does not have to continue further down in this tree.

Every node in the tree contains the number of bits of the field in the incoming packet that has to be compared and also what these bits should be to get a match. When there is a match, the field of the incoming packet is shifted this given number of bits and the next bit of the incoming packet decides which one of the two children nodes to visit next. This bit is shifted, and a new comparison is made in the corresponding child node.

Since the best matching filter is the longest match, the search must continue with the ambition to find a longer match even if a reservation already has been found. A match is better than a previous match if it contains a longer address prefix or if it comprises more fields. However, not only the number of fields in the filter decides if it is a better match, but also the priority order. If we have the same priority order as mentioned before (SA, DA, SP, DP, TOS, PROT), the filter (SP, DP) would give a better match than (SP, TOS, PROT) although it comprises fewer fields, on account of the fact that DP has higher priority than both TOS and PROT. This assumes of course that both these filters match the incoming packet. Accordingly, when we find a matching filter in a node, the search continues either

further down the trie tree or, if that is not possible, in the following binomial trees starting at that particular node.

In figure 5 and figure 6 we illustrate how insertion of filters in these trie trees works. We exemplify using fields of maximum four bits where insertion of prefixes is possible. Figure 5 illustrates the insertion of the fields 0100 and 010x, where x denotes that this bit is not included in the filter. In each node the number of shifts to perform, the shift result and if a reservation exists for this node is stored.

Figure 6 illustrates the insertion of 0100 and 010x as in the previous example, but also 01xx, 1000 and 10++01.

Memory enhancements

Each filter that is kept in our data structure is stored as a number of cells. Because of the structure of our algorithm these cells are rather many, approximately ten, per filter. They are also rather small. These cells are smaller then the smallest size of memory the kernel of NetBSD can allocate, which means that we are wasting memory every time we make a filter. To cope with this and to save time when inserting new filters a new memory allocation structure has been implemented. This new structure merges a number of small cells into larger cells which are allocated when needed. It also removes larger cells when no smaller cell in it is being used.

In the prior art section some fields of application for packet classifiers were enumerated. Our invention have the greatest advantages for mainly two of these; packet classifiers used in firewalls and packet classifiers used in a QoS context. What these two fields of application have in

common is foremost that the number of stored filters are often large and classification needs to be done fast. This means that the packet classifier used in these areas need to scale well regarding time when many filters are inserted. This is what our invention is intended for.

Our packet classifier in firewalls

A classifier is a natural and indispensable part of a firewall. Greater demands are raised on the firewalls as they become more advanced and protect larger areas. The performance must stay the same despite that more filters are stored. Newly performed tests have show that performance is one of the weakest points of today's firewalls. In the test, virtual clients instead of real clients were used. When the number of virtual clients grows beyond 48, the performance rapidly drops for many of the tested firewalls. A large share of this increase in lookup time is likely to depend on increased costs of packet classification. To meet these new demands, our packet classifier tries to improve the scaling properties regarding the lookup time of the packet classifiers data structure.

Our packet classifier already works as a packet filter. If our packet classifier should work as a circuit level gateway, extensions like session memory also needs to be implemented. A number of filters and their according actions must also be implemented to make the firewall know what to do with the packets after the classification. The most common actions are 'accept', 'deny' and 'deny and report'. An accept-filter lets all matching packets through. A deny-filter denies all matching packets and a deny-and-report filter denies matching packets and reports them too. A module to take care of the reaction on these filters must also be implemented, but this is a question of minor adjustments to the present code.

Our packet classifier in a QoS context

No matter which model of QoS that is introduced in the network, packet classifiers will be needed. These packet classifiers will often work with large amounts of flows and large amounts of reservations. To deal with this in a swift way, new packet classifiers with better performance must be developed. It is in this perspective our packet classifier is interesting.

In a QoS enabled network, packets must be compared to stored reservations to find out which class of service a packet shall be given. If this is done in every involved node as in RSVP, or if it is done at the border routers of the networks like in many DiffServ proposals, does not matter for the packet classifier. The same fields in the packet header will be of interest and the number of filters will in both cases be large.

Another application for our packet classifier in a QoS context is as a policing unit. A policing unit, as has been mentioned before, discriminates flows that send to much data. This can be realized with a packet classifier and new modules that remembers how much data each flow has sent.

The policing unit compares all incoming packets against the stored filters. If a packet matches a filter and the according reservation the flow is examined to see if it keeps to what is agreed or if it sends more. If more packets than agreed on are sent, packets are either degraded to best effort or simply dropped. In this case as well, it is demanded of the packet classifier to be able to compare incoming packets fast to a large amount of filters.

The invention is to be limited only by the scope of the claim so the description is not to be taken as limiting the invention.

Claims

1. A communicationsnetwork or a IP-network which incorporates a packet classifier in a end system or a router which inspects each incomming packet based on information in the packet headers and determines how each packet should be treated, **characterised in** that said packet classifier divides packet into classes when a large number of filters are stored.

2. A communicationsnetwork or a IP-network as claimed in claim 1, with a packet classifier that uses binomial trees and trie trees to improve the performance of the data structure.

3. A communicationsnetwork or a IP-network as claimed in claim 2, with a packet classifier that is implemented in the kernel of the operating system.

4. A communicationsnetwork or a IP-network as claimed in claim 1, with a packet classifier that can be used in firewalls an in Qos context.

5. A communicationsnetwork or a IP-network as claimed in previous claim that uses a nonlinear structure for storing filters.

6. A communicationsnetwork or a IP-network as claimed in previous claim that the packet classifier consists of three more or less detached parts which are: the reservation program 1 which is a standalone program, the interface between the packet flows 2 and the data structure 3 and the data structure itself and the main part, the data structure 3 and the interface 4 between it and the packets, is

implemented as a part of the kernel 5 in the operating system NetB.

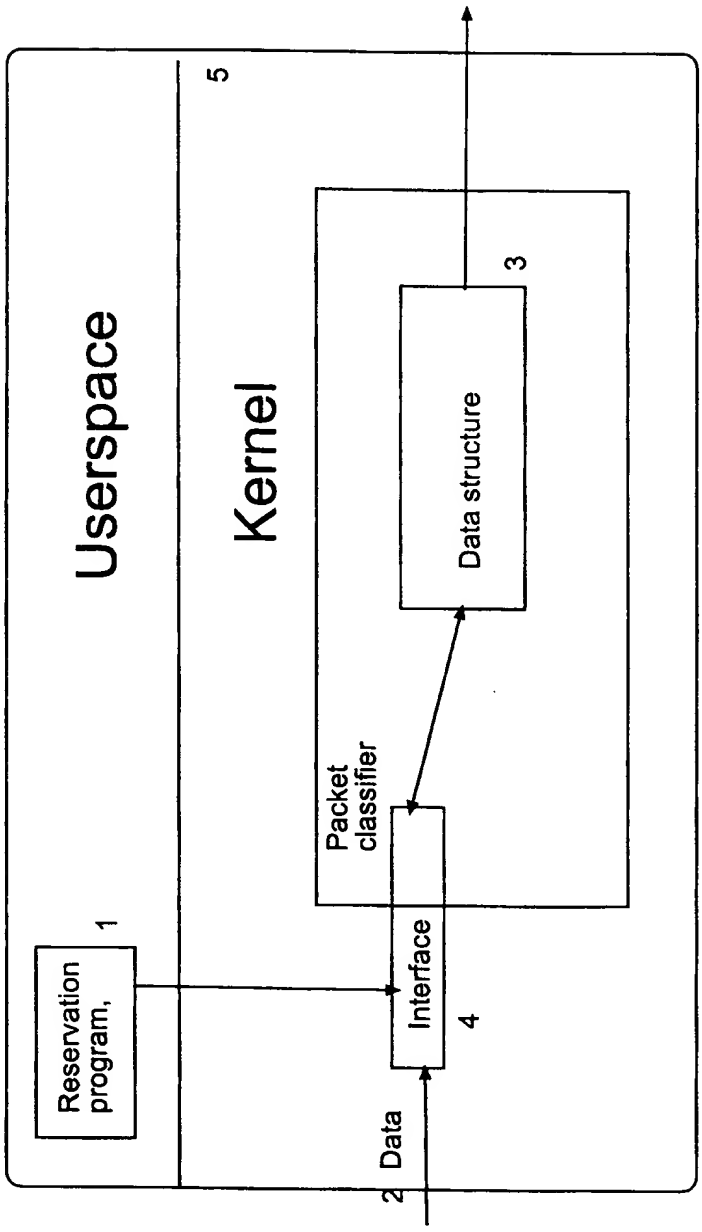

Userspace
Kernel
5
Data structure
3
Packet classifier
Interface
4
Reservation program,
1
Data
2

Fig 1

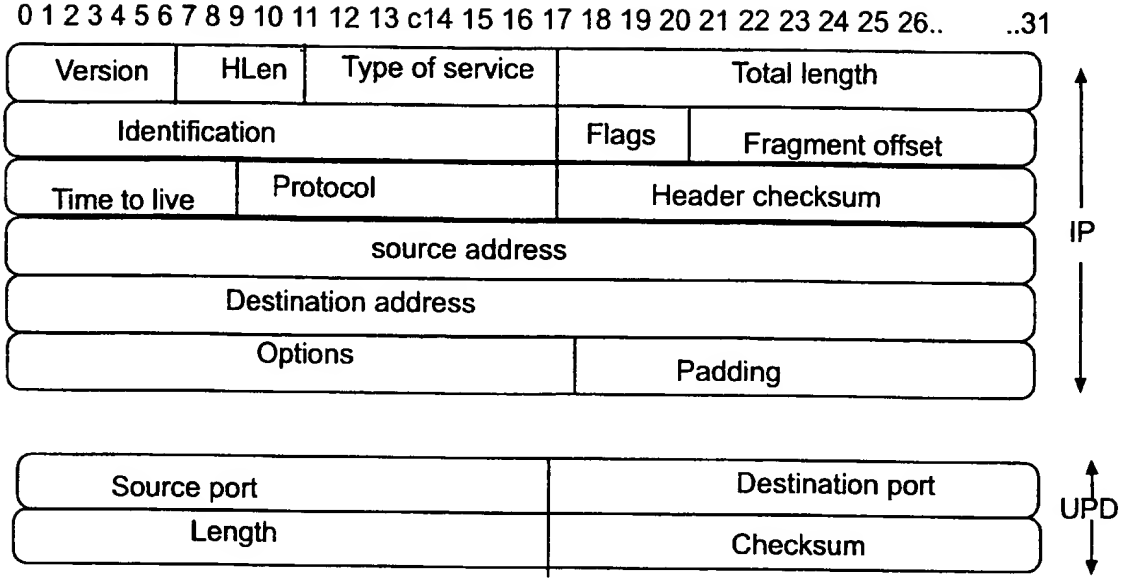

0 1 2 3 4 5 6 7 8 9 10 11 12 13 c14 15 16 17 18 19 20 21 22 23 24 25 26.. ..31
Version
HLen
Type of service
Total length
Identification
Flags
Fragment offset
Time to live
Protocol
Header checksum
source address
Destination address
Options
Padding
IP
Source port
Destination port
Length
Checksum
UPD

Fig 2

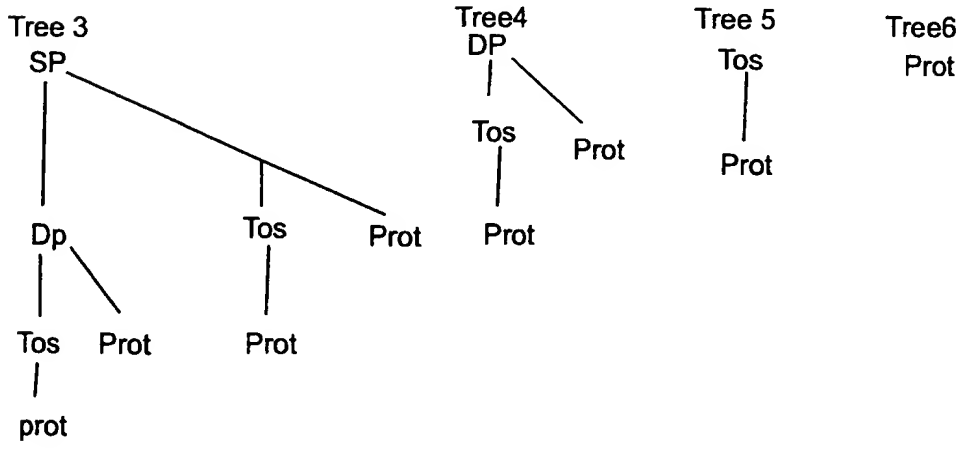

Tree 3
SP
Dp
Tos
Prot
Tos
Prot
prot
Tos
Prot
Prot
Tree4
DP
Tos
Prot
Prot
Tree 5
Tos
Prot
Tree6
Prot

Fig 3

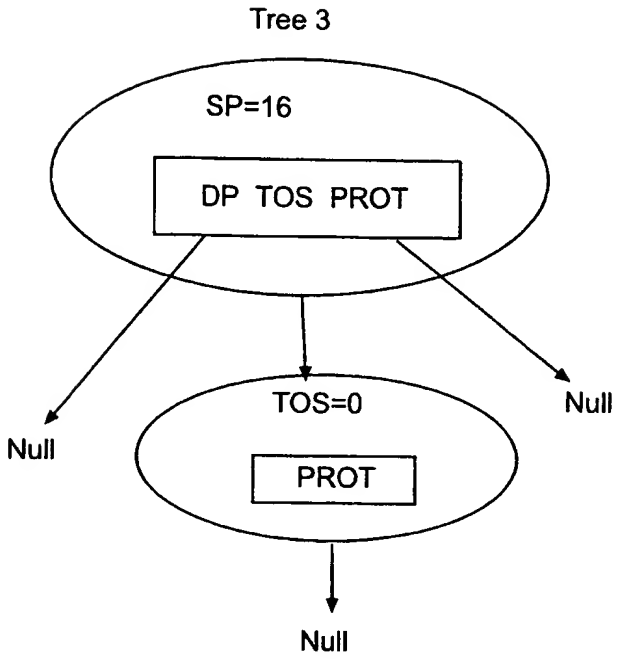

Tree 3
SP=16
DP TOS PROT
Null
TOS=0
PROT
Null
Null

Fig 4

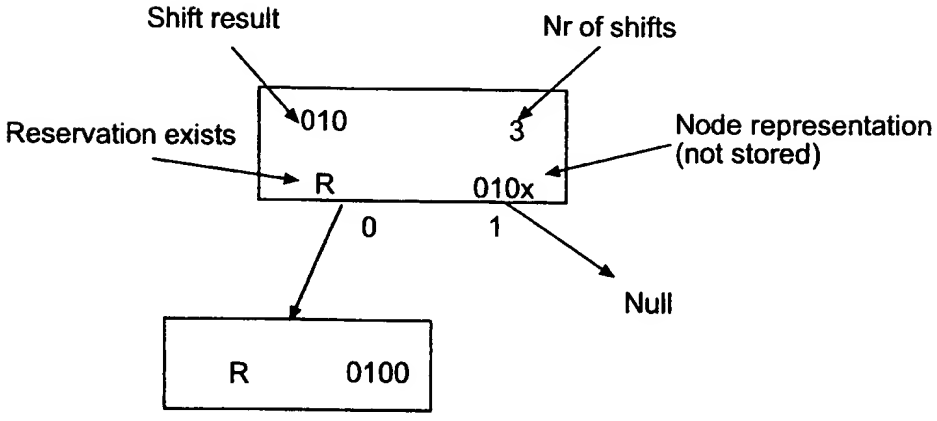

Shift result
Nr of shifts
Reservation exists
010
3
R
010x
Node representation
(not stored)
0
1
Null
R
0100

Fig 5

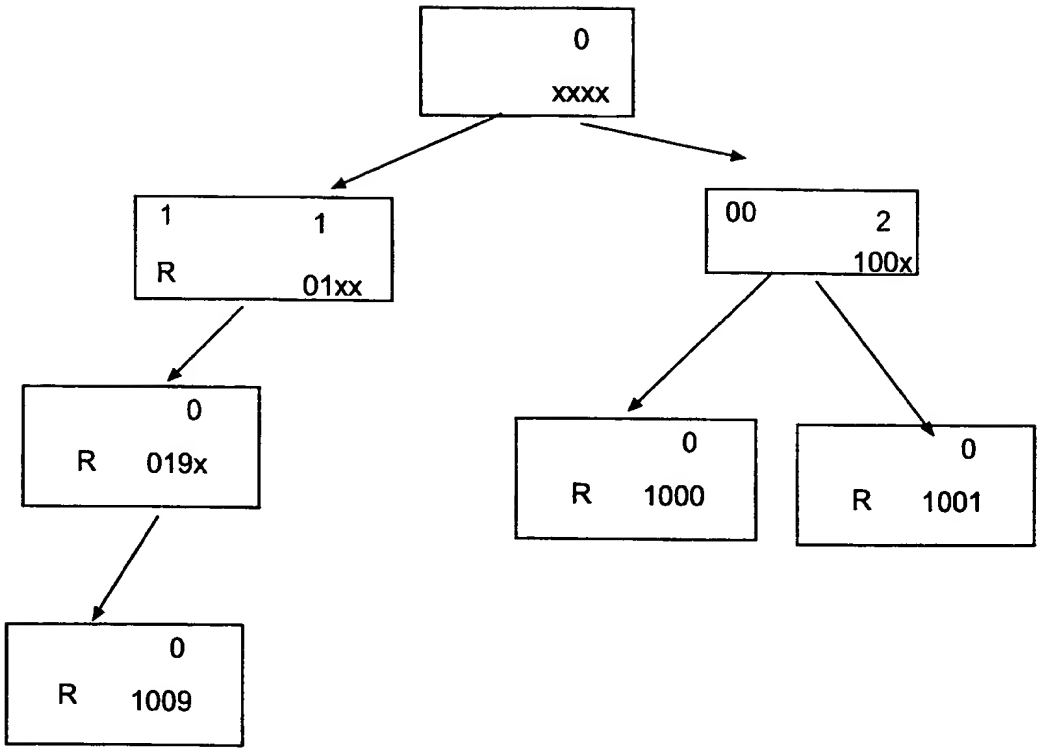



Fig 6